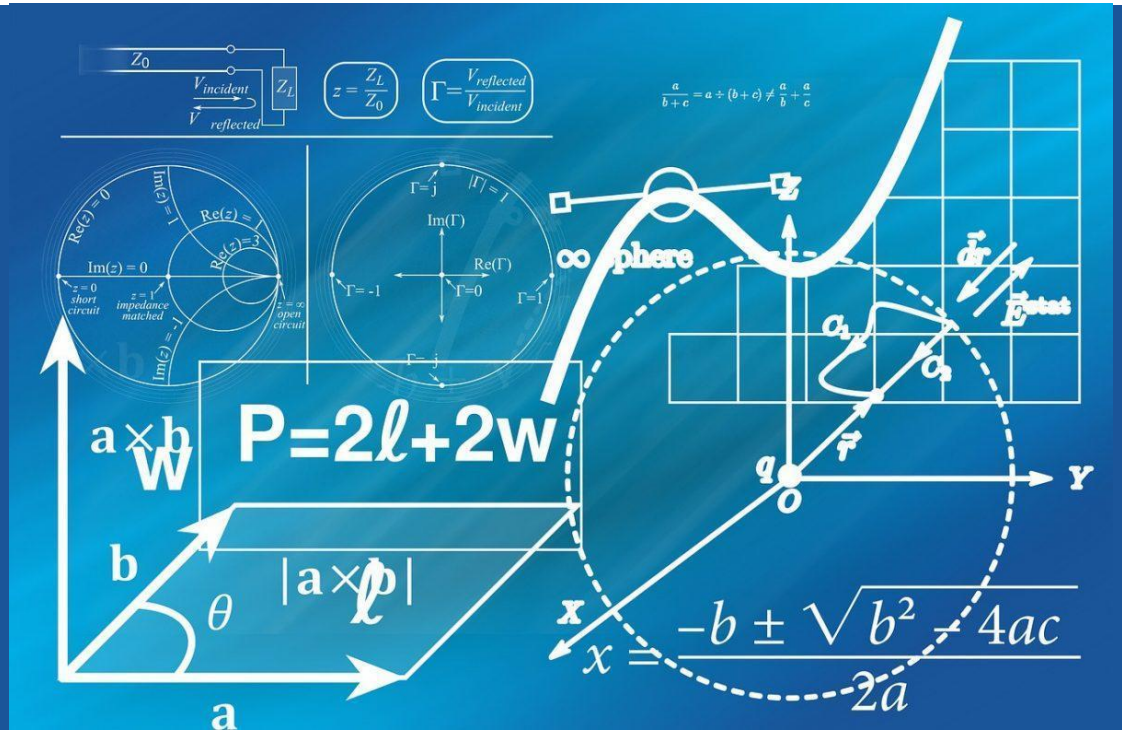
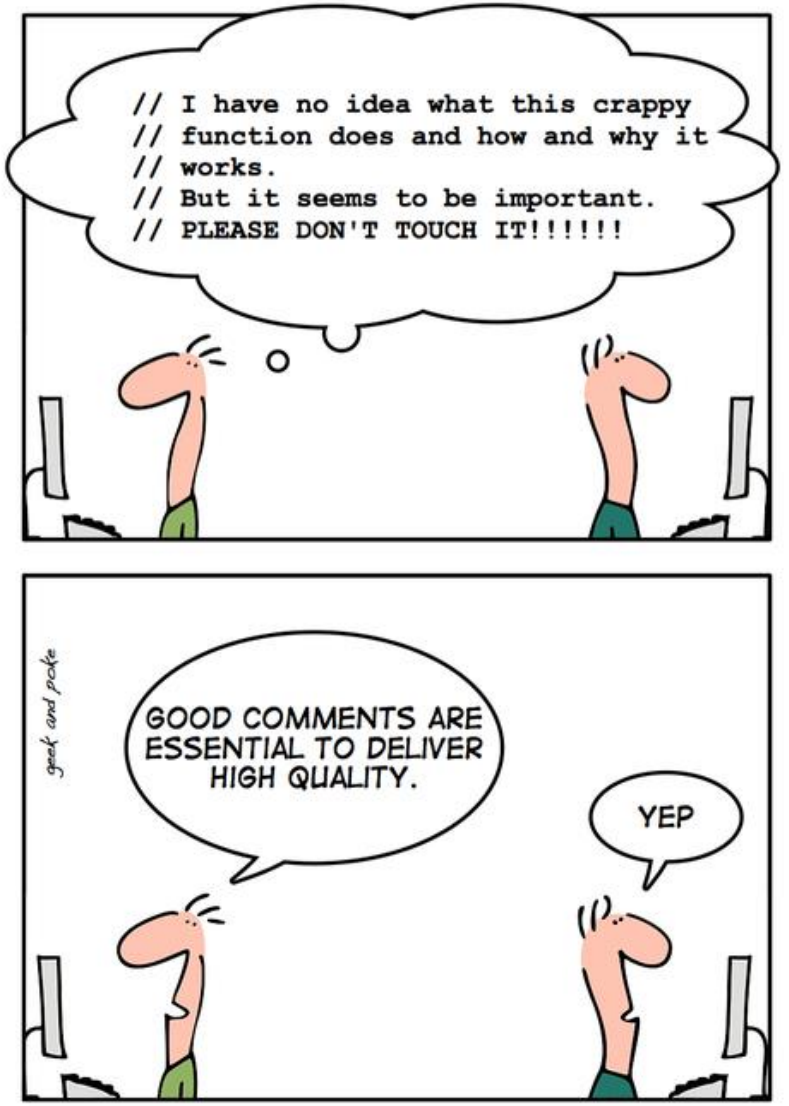
Les fonctions



Antoine SCHMID, CFPT-Informatique
Avec la collaboration du CPLN

Table des Matières

- Résumé de l’épisode précédent
- Les fonctions
  - Rôle
  - Découverte
  - Avantages
  - Valeur de retour
  - Paramètres
- Variable locales
- Les fonctions anonymes
- Bonnes pratiques pour les fonctions



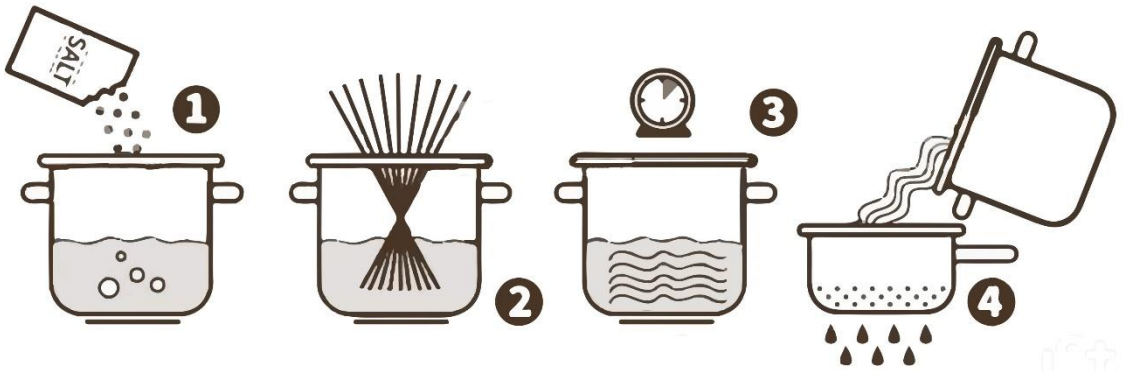
Dans l’épisode précédent…

- ```
  for (initialisation; condition; étape) {
    // …
  }
  ```

- ```
  while (condition) {
    // …
  }
  ```

- ```
  do {
    // …
  } while (condition);
  ```
  ```

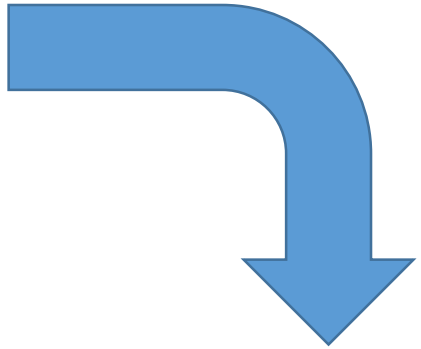
Flashback…




```
Début
    Sortir une casserole
    Mettre de l'eau dans la casserole
    Ajouter du sel
    Mettre la casserole sur le feu
    Tant que l'eau ne bout pas
      Attendre
    Sortir les pâtes du placard
    Verser les pâtes dans la casserole
    Tant que les pâtes ne sont pas cuites
        Attendre
    Verser les pâtes dans une passoire
    Remuer la passoire pour faire couler l’eau
    Verser les pâtes dans un plat
    Goûter
    Tant que les pâtes sont trop fades
        Ajouter du sel
        Goûter
    Si on préfère le beurre à l'huile
        Ajouter du beurre
    Sinon
        Ajouter de l'huile
Fin
```

```
Début
    Sortir une casserole
    Mettre de l'eau dans la casserole
    Ajouter du sel
    Mettre la casserole sur le feu
    Tant que l'eau ne bout pas
      Attendre
    Sortir les pâtes du placard
    Verser les pâtes dans la casserole
    Tant que les pâtes ne sont pas cuites
        Attendre
    Verser les pâtes dans une passoire
    Remuer la passoire pour faire couler l’eau
    Verser les pâtes dans un plat
    Goûter
    Tant que les pâtes sont trop fades
        Ajouter du sel
        Goûter
    Si on préfère le beurre à l'huile
        Ajouter du beurre
    Sinon
        Ajouter de l'huile
Fin
```

La première version détaille toutes les actions individuelles à réaliser.



La seconde décompose la recette en sous-étapes regroupant plusieurs actions individuelles

```
Début
    Faire bouillir de l’eau
    Cuire les pâtes dans l’eau
    Egoutter les pâtes
    Assaisonner les pâtes
Fin
```

Le rôle des fonctions

```
Début
    Faire bouillir de l’eau
    Cuire les pâtes dans l’eau
    Egoutter les pâtes
    Assaisonner les pâtes
Fin
```

- Cette version est plus concise et plus facile à interpréter
- Mais elle introduit des concepts relatifs au domaine de la cuisine comme cuire, égoutter, ou assaisonner.
- On peut envisager de réutiliser ces concepts pour réaliser d'autres recettes, par exemple la préparation d'un plat de riz.

Découverte des fonctions

- Une **fonction** est un regroupement d'instructions qui réalise une tâche donnée.

```
function direBonjour() {
  console.log("Bonjour !");
}

console.log("Début du programme");
direBonjour();
console.log("Fin du programme");
```

Déclaration d’une fonction

- L'opération de création d'une fonction s'appelle la **déclaration**

```
// Déclaration d'une fonction nommée maFonction
function maFonction() {
  // Instructions de la fonction
}
```

- Mot-clé **`function`** + nom de la fonction + une paire de parenthèses ().
- Les instructions qui composent la fonction constituent le **corps** de la fonction.
- Ces instructions sont placées entre accolades et indentées.

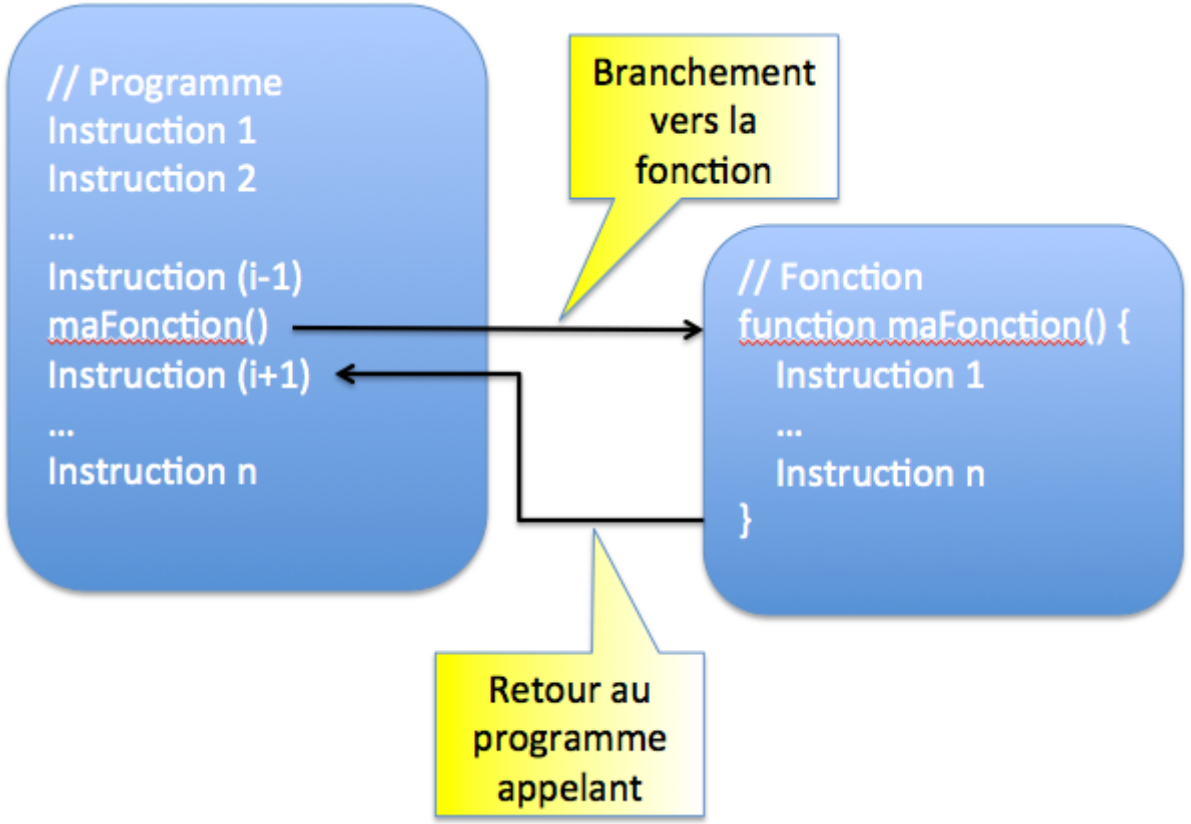
Appel d’une fonction

- **L'appel** d'une fonction s'effectue en écrivant le nom de la fonction suivi d'une paire de parenthèses.

```
// …
maFonction(); // Appel de la fonction maFonction
// …
```

- L'appel d'une fonction déclenche l'exécution des instructions qui la constituent
- Puis l'exécution reprend à l'endroit où la fonction a été appelée

Appel d’une fonction



Avantages des fonctions

- Décomposer un problème complexe en sous-problèmes plus simples.

- Le programme est plus lisible et plus facile à faire évoluer

- Evite la duplication du code

  => Principe **DRY**
  Don’t Repeat Yourself !

Valeur de retour

```
function direBonjour() {
  return "Bonjour !";
}
console.log("Début du programme");
const resultat = direBonjour();
console.log(resultat);
console.log("Fin du programme");
```

- Mot-clé **`return`** permet de donner une **valeur de retour**.
- La fonction devient une expression qui est évaluée et produit un résultat

Valeur de retour

```
function direBonjour() {
  return "Bonjour !";
}

const resultat = direBonjour();
console.log(resultat);

console.log(direBonjour());
```

- On peut sauver la valeur de retour dans une variable ou l'utiliser directement dans une instruction ou une condition

Valeur de retour

```
function direBonjour() {
  return "Bonjour !";
  // Ceci ne sera jamais exécuté
  console.log("Fin de la fonction");
}
```

- **`return`** implique la fin de la fonction
- Les instructions qui se trouvent après un return ne seront jamais exécutées

Valeur de retour – attention !

```
function maFonction() {
  // Pas d'instruction return
}
let resultat = mafonction();
console.log(resultat); // undefined
```

- Si on essaie de récupérer la valeur de retour d'une fonction qui n'inclus pas d'instruction **`return`**, on obtient la valeur **`undefined`**

- Une fonction qui ne renvoie pas de valeur s'appelle une **procédure**

Passage de paramètres

- Un **paramètre** est une information dont une fonction a besoin pour jouer son rôle

- Les paramètres d'une fonction sont définis entre parenthèses juste après le nom de la fonction.

- On peut ensuite utiliser leur valeur dans le corps de la fonction.

AVERTISSEMENT

TOUTE RESSEMBLANCE AVEC DES PERSONNES OU DES SITUATIONS EXISTANTES OU AYANT EXISTÉ NE SAURAIT ÊTRE QUE FORTUITE

Passage de paramètres

```
function direBonjour(prenom) {
  const MESSAGE = `Bonjour, ${prenom} !`;
  return MESSAGE;
}

console.log(direBonjour("Svenja")); // Bonjour, Svenja !
console.log(direBonjour("Rahmat")); // Bonjour, Rahmat !
```

- La valeur d'un paramètre est fournie au moment de l'appel de la fonction : on dit que cette valeur est **passée en paramètre**.
- On appelle **argument** la valeur donnée à un paramètre lors d'un appel.

Passage de paramètres – syntaxe

```
// Déclaration de la fonction maFonction
function maFonction(param1, param2, ...) {
  // Instructions pouvant utiliser param1, param2, ...
}

// Appel de la fonction maFonction
// param1 reçoit la valeur de arg1, param2 la valeur de arg2, ...
maFonction(arg1, arg2, ...);
```

Passage de paramètres – attention !

- Le nombre et l'ordre des paramètres doivent être respectés !

```
function presentation(prenom, age) {
  console.log(`Tu t'appelles ${prenom} et tu as ${age} ans`)
}

presentation("David", 26); // Tu t'appelles David et tu as 26 ans
presentation(26, "Roberto"); // Tu t'appelles 26 et tu as Roberto ans
```

Les variables locales

Variables locales

```
function direBonjour() {
  const MESSAGE = "Bonjour !";
  return MESSAGE;
}

console.log(direBonjour());
```

- Les variables déclarées dans le corps d'une fonction sont appelées des **variables locales**

- Elles ne sont utilisable qu'à l'intérieur de la fonction

Variables locales

```
function direBonjour() {
  const MESSAGE = "Bonjour !";
  return MESSAGE;
}

console.log(direBonjour());
console.log(MESSAGE); // Erreur : la variable MESSAGE n'existe pas
```

- Rappel : la **portée d'une variable** = l'ensemble des endroits où elle est accessible
- Pour les fonctions, la portée = le corps de la fonction

Fonctions anonymes

Fonctions anonymes

- Une fonction **anonyme** n’a pas de nom !
- On les utilise lorsqu’on n’a pas besoin d’appeler notre fonction par son nom => lorsque le code de notre fonction n’est appelé qu’à un endroit

- Elles sont souvent utilisées lors du déclenchement d’un évènement (que nous étudierons un peu plus tard)

Fonctions anonymes

```
const bonjour = function(prenom) {
  const MESSAGE = `Bonjour, ${prenom} !`;
  return MESSAGE;
}

console.log(bonjour("Loïc")); // Bonjour, Loïc !
```

- La fonction ici est **anonyme** et directement affectée à la variable bonjour.
- La valeur de cette variable est donc une fonction.
- Cette manière de créer une fonction est appelée **expression de fonction**

Fonctions anonymes – syntaxe

```
// Affectation d'une fonction anonyme à la variable maVariable
const maVariable = function(param1, param2) {
  // Instructions pouvant utiliser param1, param2, ...
}

// Appel de la fonction anonyme
// param1 reçoit la valeur de arg1, param2, la valeur de arg2, ...
maVariable(arg1, arg2, ...);
```

Fonctions anonymes – fonction fléchée

```
const bonjour = (prenom) => {
  const MESSAGE = `Bonjour, ${prenom} !`;
  return MESSAGE;
}

console.log(bonjour("Steven")); // Bonjour, Steven !
```

- Cette syntaxe est appelée **fonction fléchée** (*fat arrow function*)
- C’est juste une autre façon plus concise de créer une fonction anonyme

Fonctions anonymes – encore plus court

- Lorsque le corps de la fonction se limite à une seule ligne, on peut écrire son résultat sans créer de blocs de code avec des accolades. Dans ce cas, l'instruction **return** est implicite.
- Lorsque la fonction n'a qu'un seul argument, on peut omettre les parenthèses autour de celui-ci.

```
const bonjour = prenom => `Bonjour, ${prenom}`;

console.log(bonjour("Valentin")); // Bonjour, Valentin !
```

Bonnes pratiques pour les fonctions

1. Bien nommer ses fonctions

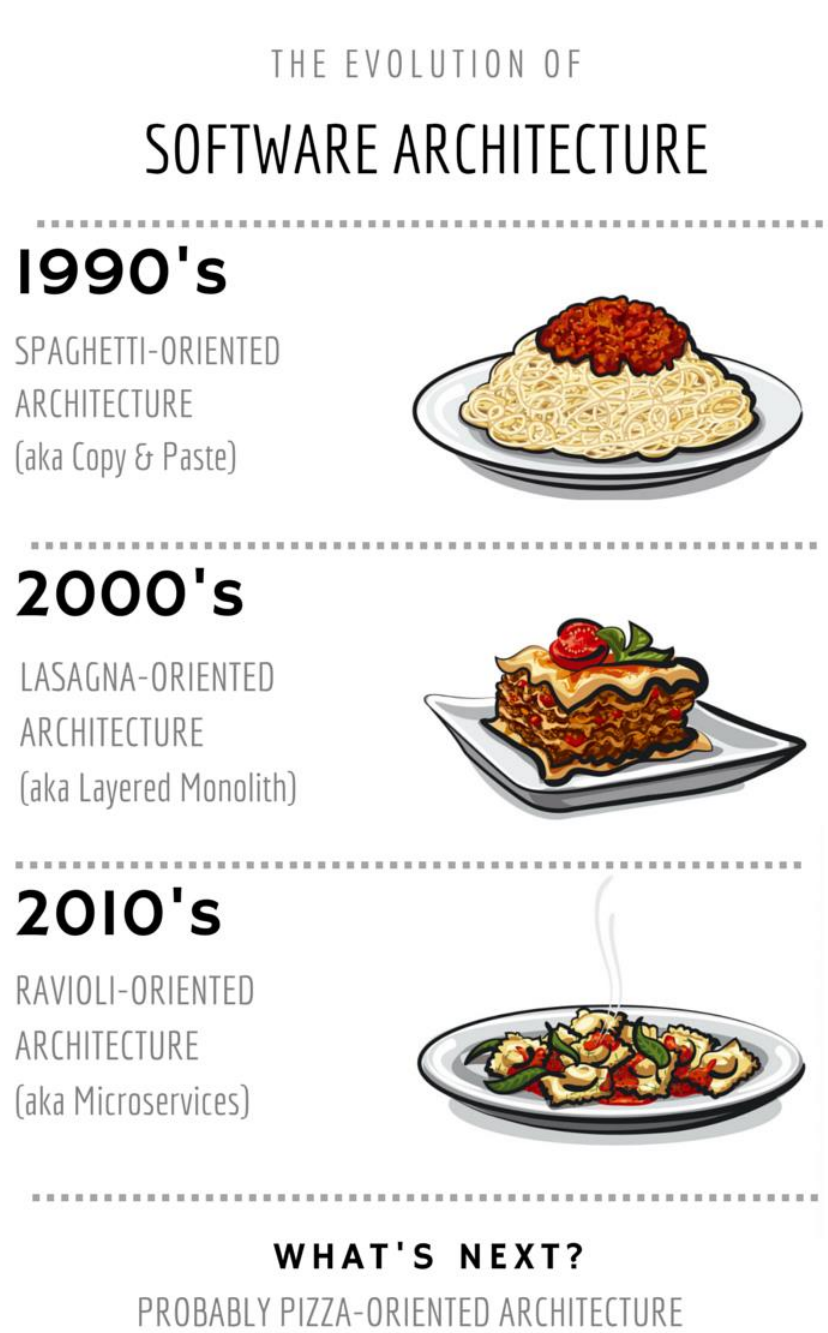
- Tout comme pour les variables, il est primordial de bien nommer ses fonctions

- Utilisez la norme lowerCamelCase
- Donnez des noms significatifs qui expriment le rôle
  - Une fonction devrait toujours être un verbe
  - Avec des mots pour spécifier les données

- Exemples
  - afficherHeure();
  - calculerPerimetre(rayon);
  - avancer(distance);

2. Limiter la complexité

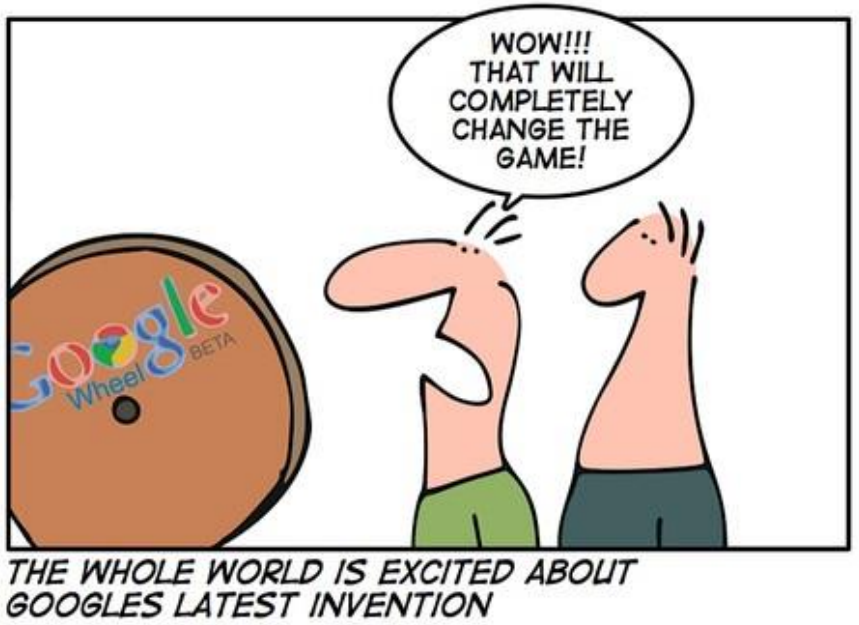
- Une fonction ne doit avec qu'un seul rôle !
- Evitez de faire des fonctions très longues (limitez le nombre de lignes)
- Ne faites des fonctions que si c'est nécessaire (DRY)



3. Ne pas réinventer la roue

- Le langage vous propose un nombre important de fonctions qui répondent à des besoins variés

- Utilisez-les
  - `prompt()`
  - `alert()`
  - `Math.min()`
  - `Math.random()`
  - …



Exercices

- Exercices 6.x : **Fonctions**